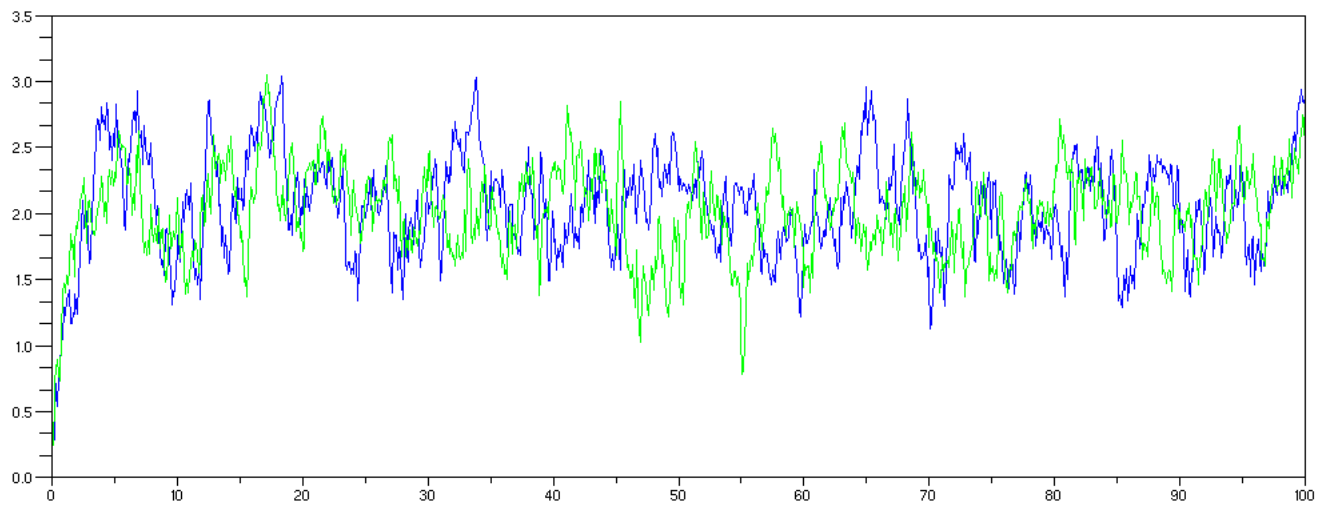


Méthodes numériques pour la finance

Rapport de TD n°2

Master SAF 2 - ISFA

5 mars 2007



Bourges Mathieu
Dutang Christophe
Sibilleau Michaël

1 Exercice n°1 (processus d'Ohrstein-Uhlenbeck)

1.1 Question a

On considère un actif risqué dont le cours est modélisé par l'équation différentielle stochastique (processus d'Ohrstein-Uhlenbeck) :

$$dS_t = \alpha(b - S_t)dt + \sigma dW_t \quad (1)$$

où b , α , σ sont des constantes positives et où S_0 est supposé connu.

Pour résoudre (1), on peut définir un premier changement de variable : $X_t = S_t - b$ et on obtient alors :

$$dX_t = -\alpha X_t dt + \sigma dW_t \quad (2)$$

puis on effectue un second changement de variable : $Y_t = X_t e^{\alpha t}$ et on utilise la formule d'Itô pour calculer la différentielle de Y_t .

Pour ce faire, on utilise la fonction indéfiniment dérivable en espace et en temps : $f(t, x) = x e^{\alpha t}$, il vient :

$$dY_t = X_t \alpha e^{\alpha t} + e^{\alpha t} dX_t + 0$$

$$\text{soit d'après (2) : } dY_t = X_t \alpha e^{\alpha t} + e^{\alpha t} (-\alpha X_t dt + \sigma dW_t) = \sigma e^{\alpha t} dW_t$$

$$\text{Par intégration, on obtient alors : } Y_t = Y_0 + \int_0^t \sigma e^{\alpha s} dW_s$$

En revenant à la variable $X_t = Y_t e^{-\alpha t}$ et en remarquant que : $Y_0 = X_0$, on a :

$$X_t = e^{-\alpha t} X_0 + e^{-\alpha t} \int_0^t \sigma e^{\alpha s} dW_s$$

ce qui permet d'obtenir (compte tenu que $X_t = S_t - b$) :

$$S_t = e^{-\alpha t} S_0 + b(1 - e^{-\alpha t}) + e^{-\alpha t} \sigma \int_0^t e^{\alpha s} dW_s \quad (3)$$

□

1.2 Question b

L'expression (3) montre que le processus S_t est gaussien et à T fixé, S_T suit une loi normale dont il reste à calculer les deux premiers moments.

On sait que $e^{-\alpha t} \sigma \int_0^T e^{\alpha s} dW_s$ est de loi normale centrée donc : $\mathbb{E}(S_T) = e^{-\alpha T} S_0 + b(1 - e^{-\alpha T})$

Calculons maintenant la variance :

$$\begin{aligned}
 \text{Var}(S_T) &= \mathbb{E} \left[(S_T - \mathbb{E}(S_T))^2 \right] \\
 &= \mathbb{E} \left[\left(e^{-\alpha t} \sigma \int_0^T e^{\alpha s} dW_s \right)^2 \right] \\
 &= e^{-2\alpha t} \sigma^2 \mathbb{E} \left[\left(\int_0^T e^{\alpha s} dW_s \right)^2 \right] \\
 &= e^{-2\alpha t} \sigma^2 \int_0^T e^{2\alpha s} ds \\
 &= e^{-2\alpha t} \sigma^2 \left[\frac{e^{2\alpha s}}{2\alpha} \right]_0^T \\
 &= \sigma^2 \frac{1 - e^{-2\alpha T}}{2\alpha}
 \end{aligned}$$

□

Remarque : Comme α est positif, on observe que $\lim_{T \rightarrow +\infty} \mathbb{E}(S_T) = b$ et que $\lim_{T \rightarrow +\infty} \text{Var}(S_T) = \frac{\sigma^2}{2\alpha}$

Lorsque T tend vers l'infini, S_T tend donc vers une loi normale de moyenne b et de variance $\frac{\sigma^2}{2\alpha}$. C'est la propriété de retour à la moyenne du processus d'Ohrstein-Uhlenbeck.

Pour implémenter une fonction qui renvoie un échantillon de taille n de même loi que S_T , il suffit donc tout simplement de simuler une variable aléatoire normale d'espérance $\mathbb{E}(S_T)$ et de variance $\text{Var}(S_T)$ avec les expressions obtenues *supra*.

L'histogramme de l'échantillon obtenu est donné sur la figure 1 . Il illustre bien l'allure d'une loi normale.

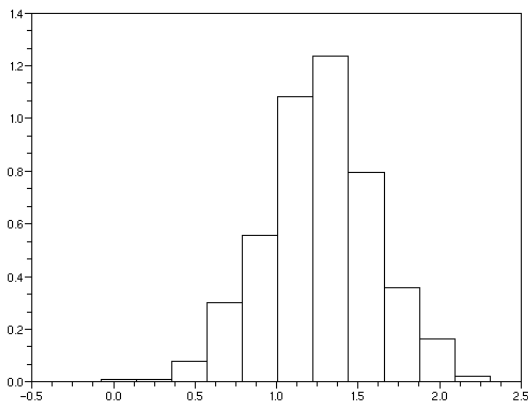


FIG. 1 – histogramme d'un échantillon de taille 1.000 de même loi que S_T

1.3 Question c

Soit $X \sim N(\mu, \sigma^2)$ de densité f . On se propose de calculer la quantité $\mathbb{E}[\max(0, X)]$

On a :

$$\begin{aligned} \mathbb{E}[\max(0, X)] &= \int_{\mathbb{R}} \max(x, 0) f(x) dx \\ &= \int_0^{+\infty} x f(x) dx \\ &= \int_0^{+\infty} (x - \mu) f(x) dx + \mu \int_0^{+\infty} f(x) dx \end{aligned}$$

On effectue pour chaque membre le changement de variable $t := \frac{x-\mu}{\sigma}$, il vient :

$$\begin{aligned} \mathbb{E}[\max(0, X)] &= \frac{\sigma}{\sqrt{2\pi}} \int_{-\mu/\sigma}^{+\infty} t e^{-t^2/2} dt + \frac{\mu}{\sqrt{2\pi}} \int_{-\mu/\sigma}^{+\infty} e^{-t^2/2} dt \\ &= -\frac{\sigma}{\sqrt{2\pi}} \left[e^{-t^2/2} \right]_{-\mu/\sigma}^{+\infty} + \mu [\Phi(t)]_{-\mu/\sigma}^{+\infty} \\ &= \frac{\sigma}{\sqrt{2\pi}} e^{-\mu^2/2\sigma^2} + \mu \left[1 - \Phi\left(-\frac{\mu}{\sigma}\right) \right] \end{aligned}$$

avec Φ la fonction de répartition de la loi normale centrée réduite.

□

1.4 Question d

La valeur d'un Put européen est donné par : $Put = e^{-rT} \mathbb{E}_{\mathbb{Q}}[(K - S_T)^+]$ où S est le cours du sous jacent, K le prix d'exercice, T la date d'échéance et r le taux sans risque. La question précédente nous permet donc d'obtenir une valeur théorique.

Mais on peut également utiliser les méthodes de Monte Carlo qui approchent le calcul d'espérance par une moyenne pour construire un estimateur :

$$\widetilde{Put}^n = e^{-rT} \frac{1}{n} \sum_{i=1}^n \max(K - S_{T_i}, 0)$$

La fonction Scilab *PutMonteCarlo* permet d'obtenir le graphe 2 avec les données de l'énoncé.

Cette fonction retourne également les valeurs (très proches) :

- Monte Carlo : 0.0389672
- Théorique : 0.0392587

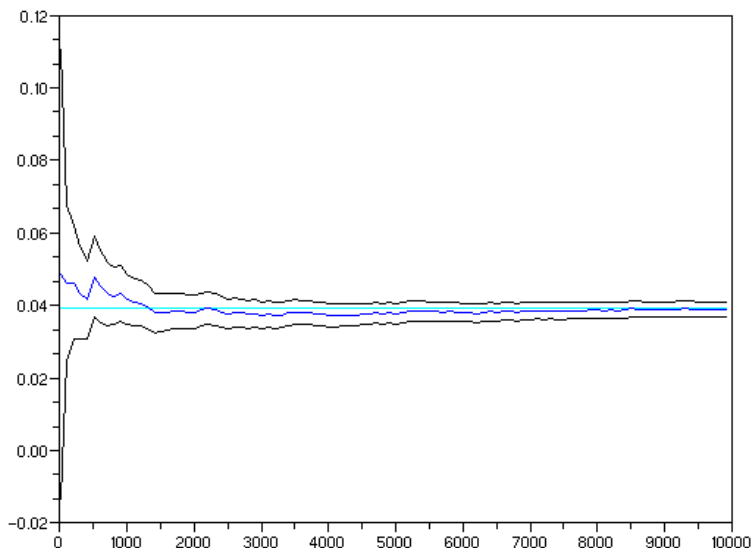


FIG. 2 – Valeur d’un Put en fonction du nombre de simulations

1.5 Question e

1.5.1 Trajectoire exacte

Reprenons les notations et la méthode de 1.1, nous savons que : $dY_t = \sigma e^{\alpha t} dW_t$ et nous intégrons cette fois entre s et t ($s \leq t$), il vient :

$$Y_t = Y_s + \int_s^t e^{\alpha u} dW_u$$

ce qui donne, avec $X_t = Y_t e^{-\alpha t}$:

$$X_t = e^{-\alpha t} Y_s + \sigma e^{-\alpha t} \int_s^t e^{\alpha u} dW_u$$

et avec : $S_t = X_t + b$:

$$S_t = b + e^{-\alpha t} Y_s + \sigma e^{-\alpha t} \int_s^t e^{\alpha u} dW_u$$

cette fois on a : $Y_s = (S_s - b) e^{\alpha s}$, ce qui conduit à :

$$\text{d'où : } S_t = e^{\alpha(s-t)} S_s + b [1 - e^{\alpha(s-t)}] + \sigma e^{-\alpha t} \int_s^t e^{\alpha u} dW_u \quad (4)$$

□

Pour obtenir un tracé de la trajectoire exacte avec (4), on utilise la fonction Scilab *TraceTrajectoire* avec un appel à la fonction *SimulSousJacTrajExact*. La figure 3 donne une illustration des résultats obtenus (deux trajectoires de couleur différente). D’autres simulations sont disponibles au format .gif dans l’archive de compte-rendu.

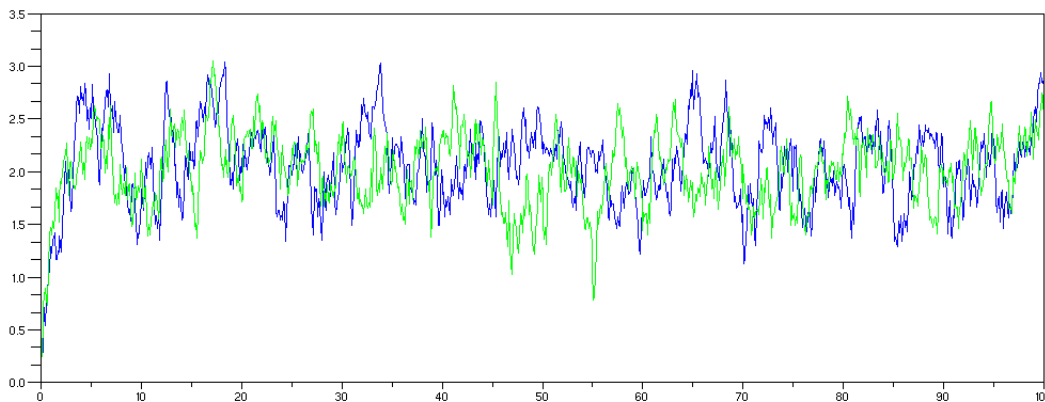


FIG. 3 – Tracé de 1.000 réalisations du processus d’Ohrstein-Uhlenbeck (2 trajectoires)

1.5.2 Schéma d’Euler

En discrétisant (1), nous obtenons :

$$\tilde{S}_{t_{i+1}} - \tilde{S}_{t_i} = \alpha \left(b - \tilde{S}_{t_i} \right) (t_{i+1} - t_i) + \sigma \left(W_{t_{i+1}} - W_{t_i} \right)$$

La fonction Scilab *SimulSousJacTrajEuler* renvoie une trajectoire approchée du processus. Quant à la fonction *TraceFdRHisto*, elle trace les fonctions de répartition empirique et théorique de S (en partie gauche) ainsi que l’histogramme de \tilde{S} (en partie droite).

Les résultats obtenus sont illustrés sur la figure 4. Les fonctions de répartition empirique sont calculées pour deux tailles d’échantillon : 100 (en vert) et 100.000 (en bleu). La fonction de répartition théorique (rouge) et l’histogramme sont calculés sur un échantillon de taille 100.000.

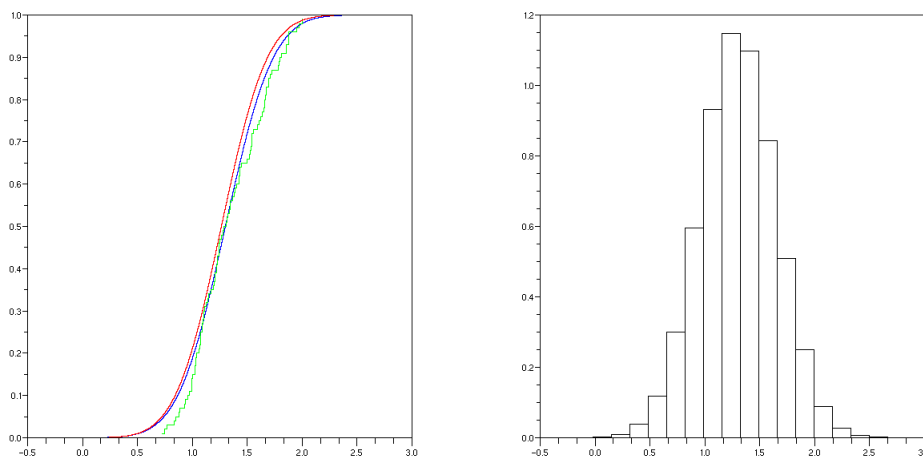


FIG. 4 – A gauche : fonctions de répartition théorique (rouge) et empiriques (bleu et vert) - A droite : histogramme de \tilde{S}

On constate une convergence rapide de la fonction de répartition de S (dès 100 simulations), ce qui permet de conclure à une bonne approximation du schéma d’Euler.

L'évaluation du Put par la méthode de Monte Carlo suivant l'approximation d'Euler s'effectue au moyen de la fonction Scilab *PutEuler* et conduit aux résultats suivants :

- Monte Carlo : 0.0368551
- Théorique : 0.0392587

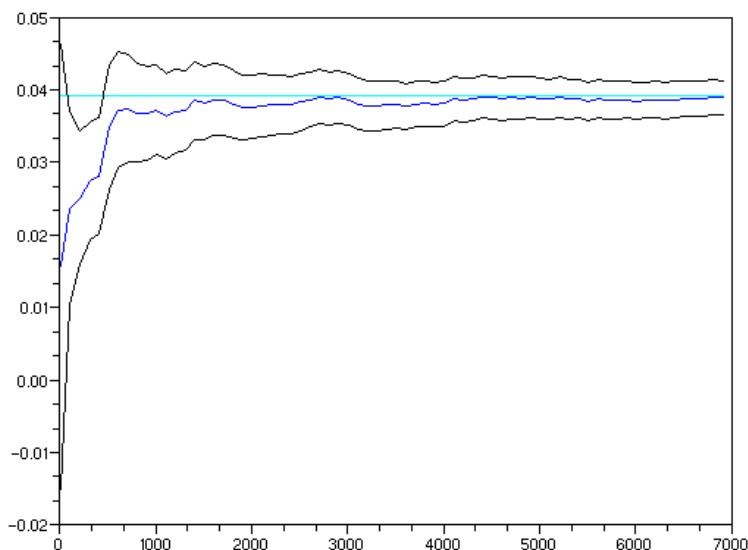


FIG. 5 – Valeur d'un Put suivant avec le schéma d'Euler en fonction du nombre de simulations

2 Exercice n°2 (modèle de Cox-Ross-Rubinstein)

2.1 Partie A

On pose dans un premier temps : $u = e^{\sigma\sqrt{h}}$, $v = e^{-\sigma\sqrt{h}}$ et $R = e^{rh} - 1$

2.1.1 Question a

Sous la probabilité risque neutre, les actifs actualisés sont des \mathcal{F}_n martingales, en particulier :

$$\mathbb{E}_{\mathbb{Q}} \left(e^{-rh} S_1^{(N)} \mid \mathcal{F}_0 \right) = S_0^{(N)} = x$$

si et seulement si : $pu + (1-p)v = e^{-rh}x$ avec : $d < 1 + R < u$

$$\text{d'où : } p = \frac{e^{-rh} - d}{u - d}$$

$$\text{ce qui s'écrit encore : } p = \frac{e^{-rh} - d}{u - d} = \frac{1 + R - d}{u - d} \quad (\text{condition risque-neutre})$$

□

2.1.2 Question b

$S_N^{(N)}$ prend les valeurs suivantes : $\{xu^k d^{N-k}\}_{0 \leq k \leq N}$ sur une probabilité de tirage avec remise, ce qui donne :

$$\forall k \in [0, N] \quad \mathbb{P}\left(S_N^{(N)} = xu^k d^{N-k}\right) = C_N^k p^k (1-p)^{N-k}$$

On en déduit que :

$$\mathbb{E}\left[f\left(S_N^{(N)}\right)\right] = \sum_{k=0}^N f(xu^k d^{N-k}) \mathbb{P}\left(S_n^{(N)} = xu^k d^{N-k}\right) = \sum_{k=0}^N C_N^k p^k (1-p)^{N-k} f\left(S_0^{(N)} u^k d^{N-k}\right) \quad (5)$$

□

Application : prenons : $f(x) = e^{-rT} (K - x)^+$, on obtient la valeur d'un Put européen de prix d'exercice K :

$$Put(S, T, K) = e^{-rT} \sum_{k=0}^N C_N^k p^k (1-p)^{N-k} \left(K - S_0^{(N)} u^k d^{N-k}\right)$$

La fonction Scilab *PutCombinatoire* calcule la valeur de ce Put européen.

2.1.3 Question c

$$\text{Notons : } V_n = \mathbb{E}\left[(1+R)^{n-N} f\left(S_N^{(N)}\right) \mid \mathcal{F}_n\right] \quad (6)$$

Calcul de V_0 pour $f(x) = x$:

$$\text{En utilisant (6) pour } f(x) = x, \text{ il vient : } V_n = \mathbb{E}\left[(1+R)^{n-N} S_N^{(N)} \mid \mathcal{F}_n\right]$$

$$\text{d'où : } V_0 = \mathbb{E}\left[\frac{1}{(1+R)^N} S_N^{(N)} \mid \mathcal{F}_0\right]$$

$(S_n^{(N)})_{n=0, \dots, N}$ étant une chaîne de Markov, \mathcal{F}_n adaptée, on peut écrire :

$$V_0 = \frac{1}{(1+R)^N} \mathbb{E}\left(S_N^{(N)} \mid S_0^{(N)}\right)$$

soit, d'après (5) :

$$V_0 = \frac{S_0^{(N)}}{(1+R)^N} \sum_{k=0}^N C_N^k (pu)^k [d(1-p)]^{N-k}$$

ce qui s'écrit encore :

$$V_0 = \frac{S_0^{(N)}}{(1+R)^N} (pu + (1-p)d)^N$$

La condition risque-neutre $pu + (1-p)d = 1 + R$ conduit à :

$$V_0 = \frac{S_0^{(N)}}{(1+R)^N} (1+R)^N = S_0^{(N)}$$

Avec la condition $f(x) = x$, la valeur du contrat est égale au cours du sous jacent, ce que montre l'égalité *supra*.

□

Relation de récurrence :

Posons : $V_n = v(n, S_n^{(N)})$ et considérons le portefeuille constitué de la vente d'un Put européen et de l'achat de k unités de sous jacent (voir tableau ci-dessous).

	date n	date $(n + 1)$, cas up	date $(n + 1)$, cas down
vente du contrat	$-v(n, S_n^{(N)})$	$-v(n + 1, uS_n^{(N)})$	$-v(n + 1, dS_n^{(N)})$
achat du sous jacent	$kS_n^{(N)}$	$kuS_n^{(N)}$	$kdS_n^{(N)}$
TOTAL	$kS_n^{(N)} - v(n, S_n^{(N)})$	$kuS_n^{(N)} - v(n + 1, uS_n^{(N)})$	$kdS_n^{(N)} - v(n + 1, dS_n^{(N)})$

TAB. 1 – Portefeuille constitué de la vente d'un Put européen et de l'achat de k unités de sous jacent

Le nombre k d'unités de sous jacent acheté doit être déterminé de sorte que le portefeuille soit sans risque i.e. :

$$kuS_n^{(N)} - v(n + 1, uS_n^{(N)}) = kdS_n^{(N)} - v(n + 1, dS_n^{(N)})$$

soit :

$$k = \frac{v(n + 1, uS_n^{(N)}) - v(n + 1, dS_n^{(N)})}{(u - d)S_n^{(N)}} \quad (7)$$

En absence d'opportunité d'arbitrage, un portefeuille sans risque doit rapporter la taux d'intérêt sans risque, ce qui donne :

$$\left[kS_n^{(N)} - v(n, S_n^{(N)}) \right] e^{rh} = kuS_n^{(N)} - v(n + 1, uS_n^{(N)})$$

En utilisant l'expression (7), on obtient :

$$\begin{aligned} v(n, S_n^{(N)}) &= e^{-rh}v(n + 1, uS_n^{(N)}) + (1 - ue^{-rh}) \left[\frac{v(n + 1, uS_n^{(N)}) - v(n + 1, dS_n^{(N)})}{u - d} \right] \\ &= e^{-rh} \left(1 + \frac{e^{rh} - u}{u - d} \right) v(n + 1, uS_n^{(N)}) + e^{-rh} \frac{u - e^{rh}}{u - d} v(n + 1, dS_n^{(N)}) \end{aligned}$$

en utilisant le fait que : $e^{rh} = 1 + R$ et que : $p = \frac{1+R-d}{u-d}$ on a :

$$v(n, S_n^{(N)}) = \frac{1}{1+R} \left[pv(n+1, uS_n^{(N)}) + (1-p)v(n+1, dS_n^{(N)}) \right]$$

Nous avons par ailleurs : $v(N, S_N^{(N)}) = f(S_N^{(N)}) = (K - S_N^{(N)})^+$ s'agissant d'un Put de prix d'exercice K

Des deux égalités précédentes, on peut conclure que v vérifie la relation de récurrence :

$$\begin{cases} v(N, x) &= f(x) \\ v(n, x) &= \frac{1}{1+R} [pv(n+1, ux) + (1-p)v(n+1, dx)] \end{cases}$$

□

La fonction *PutArbre* utilise cette relation de récurrence pour valoriser un Put européen de prix d'exercice K et de taux constant r

2.1.4 Question d

Les tracés du graphe $S_0^{(N)} \mapsto V_0$ pour $N = 100$ avec la méthode binomiale (à gauche) et la relation de récurrence (à droite) sont reportés sur la figure 6 (on y a ajouté le prix théorique Black & Scholes)

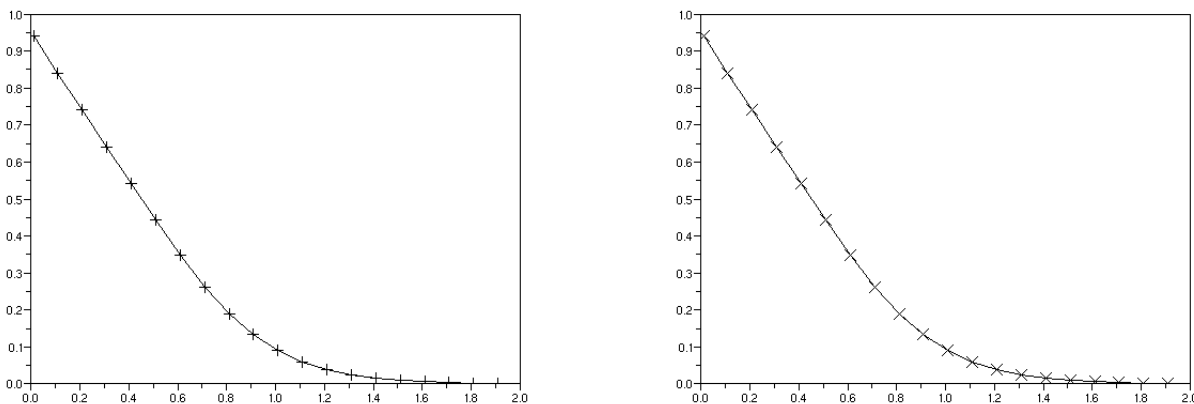


FIG. 6 – graphes $S_0^{(N)} \mapsto V_0$ pour $N=100$: méthode binomiale (à gauche) et de récurrence (à droite)

Pour $N = 100$ les résultats obtenus sont très satisfaisants (dans les deux cas), ce qui est conforme à nos attentes puisque le modèle est inchangé. Seules les méthodes de calculs numériques sont modifiées.

2.1.5 Question e

En absence d'opportunité d'arbitrage, la parité Call Put est donnée par :

$$Call(S, T, K) - Put(S, T, K) = S_0^{(N)} - Ke^{-rT} \quad (8)$$

La fonction *VerifParite* renvoie graphiquement la différence (en valeur absolue) entre les membres de gauche et de droite de l'équation (8) en fonction de $S_0^{(N)}$. On obtient la figure 7.

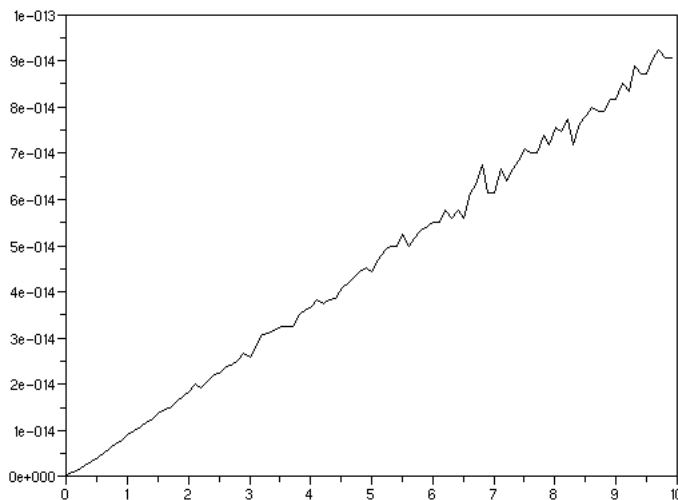


FIG. 7 – $|Call(S, T, K) - Put(S, T, K) - S_0^{(N)} + Ke^{-rT}| = f(S_0^{(N)})$

Pour un sous jacent dont le cours s'élève à 10 unités monétaires et pour un prix d'exercice de 1 on obtient un écart de l'ordre de 10^{-14} , ce qui permet de conclure à l'égalité empirique de (8).

2.1.6 Question f

Note liminaire :

Reprenons l'expression de $p = \frac{1+R-d}{u-d}$

que nous pouvons encore écrire sous la forme : $p = \frac{e^{rh} - e^{-\sigma\sqrt{h}}}{e^{\sigma\sqrt{h}} - e^{-\sigma\sqrt{h}}}$

en procédant à un développement limité à l'ordre \sqrt{h} , nous obtenons :

$$p \underset{0+}{\sim} \frac{\sigma\sqrt{h}}{2\sigma\sqrt{h}} = \frac{1}{2}$$

Le modèle de proposé par Cox-Ross-Rubinstein met en évidence que la convergence vers le modèle de Black & Scholes n'est pas un problème offrant une solution unique. En particulier, les paramètres u et d

choisis impliquent que la probabilité risque neutre ne vaut $1/2$ qu'à la limite.

Jarrow et Rudd (1983) offre une paramétrisation alternative au modèle binomial de Cox-Ross-Rubinstein. Ils proposent de fixer les paramètres comme suit :

$$u = e^{\left(r - \frac{\sigma^2}{2}\right)h + \sigma\sqrt{h}}$$

$$d = e^{\left(r - \frac{\sigma^2}{2}\right)h - \sigma\sqrt{h}}$$

et : $p = \frac{1}{2}$

La probabilité risque neutre est d'emblée fixée à $1/2$ tandis qu'en contre partie, le taux sans risque et la volatilité sont intégrés dans les paramètres u et d .

Il est important de noter qu'en théorie les branches de l'arbre ne se recombinaient pas puisque $u.d \neq 1$

Retour à l'exercice :

On applique les algorithmes précédents en considérant les paramètres de Jarrow et Rudd. Pour ce faire, on considère qu'une approximation sur la recombinaison est possible attendu que :

$$u.d = e^{2\left(r - \frac{\sigma^2}{2}\right)h}$$

et que les données numériques sont telles que : $2\left(r - \frac{\sigma^2}{2}\right)h = 10^{-4}$, ce qui implique $u.d \approx 1$

Les résultats empiriques sont les suivants :

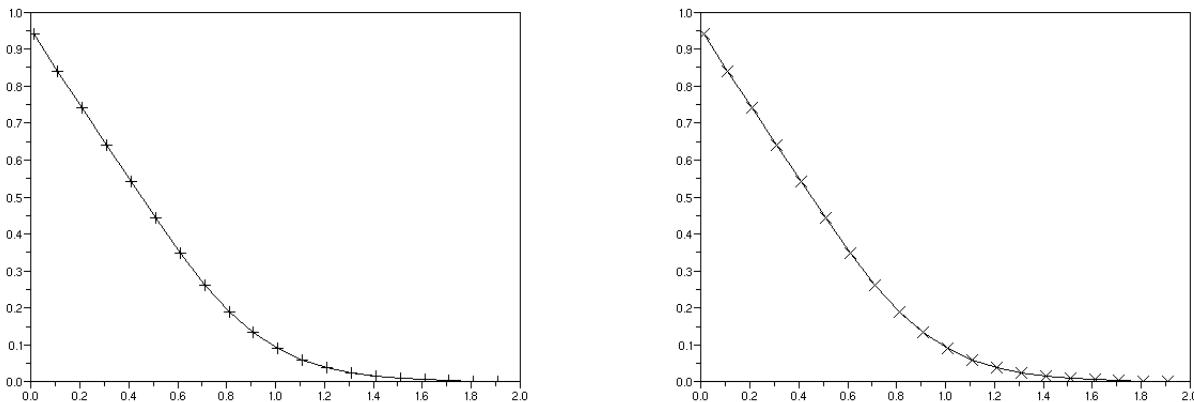


FIG. 8 – graphes $S_0^{(N)} \mapsto V_0$ pour $N=100$: méthode binomiale (à gauche) et de récurrence (à droite) - Modèle de J.R.

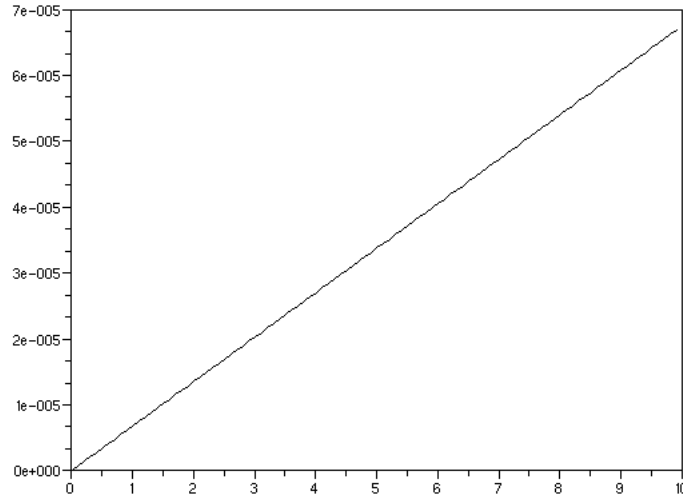


FIG. 9 – $|Call(S, T, K) - Put(S, T, K) - S_0^{(N)} + Ke^{-rT}| = f(S_0^{(N)})$ - Modèle de J.R.

Observons le graphe 9, on constate que pour un sous jacent de même valeur 10 unités monétaires, la précision de l'équation (8) est moins bonne (10^{-4} contre 10^{-14}). On pouvait s'attendre à ce résultat compte tenu des remarques faites en propos liminaires.

2.2 Partie B

2.2.1 Question a

La valeur W_{n+1} d'une option américaine de payoff $f(S_{n+1}^{(N)})$ à la date $n + 1$ est la somme d'argent dont le vendeur doit disposer à cette date pour honorer son contrat **à toute date postérieure**.

Raisonnons par récurrence descendante et supposons que la somme d'argent W_{n+1} nécessaire à la couverture de l'option entre $n + 1$ et N soit connue en $n + 1$.

W_n doit alors permettre au vendeur de :

- couvrir un éventuel exercice de l'option à la date n ce qui implique $W_n \geq f(S_n^{(N)})$ (1')
- disposer le lendemain de l'exercice de la somme W_{n+1} , ce qui peut être obtenu par réplication. On doit donc avoir $W_n \geq P_n$ où P_n est la valeur du portefeuille tel que $P_{n+1} = W_{n+1}$. Or sous la probabilité risque neutre \mathbb{Q} , les actifs actualisés sont des \mathcal{F}_n martingales, d'où : $P_n = \frac{1}{1+R} \mathbb{E}_{\mathbb{Q}}[W_{n+1} | \mathcal{F}_n]$
On en déduit donc que : $W_n \geq \frac{1}{1+R} \mathbb{E}_{\mathbb{Q}}[W_{n+1} | \mathcal{F}_n]$ (2')

Les relations (1') et (2') permettent donc d'écrire que la suite $(W_n)_{n \in [0, N]}$ définie par :

$$\begin{cases} W_N &= f(S_N^{(N)}) \\ W_n &= \text{Sup} \left[f(S_n^{(N)}), \frac{1}{1+R} \mathbb{E}(W_{n+1} | \mathcal{F}_n) \right] \end{cases}$$

définit la valeur à la date n d'une option américaine de payoff f .

2.2.2 Question b

Pour établir une relation de récurrence, nous suivons un raisonnement analogue à la question 2.1.3 : posons : $W_n = w(n, S_n^{(N)})$ et considérons le portefeuille constitué de la vente d'un Put américain et de l'achat de k unités de sous jacent (voir tableau 2).

	date n	date $(n + 1)$, cas up	date $(n + 1)$, cas down
vente contrat	$-w(n, S_n^{(N)})$	$-Sup[f(uS_n^{(N)}), w(n + 1, uS_n^{(N)})]$	$-Sup[f(dS_n^{(N)}), w(n + 1, dS_n^{(N)})]$
achat ASJ	$kS_n^{(N)}$	$kuS_n^{(N)}$	$kdS_n^{(N)}$

TAB. 2 – Portefeuille constitué de la vente d'un Put américain et de l'achat de k unités de sous jacent

Le nombre k d'unités de sous jacent acheté doit être déterminé de sorte que le portefeuille soit sans risque i.e. :

$$kuS_n^{(N)} - Sup[f(uS_n^{(N)}), w(n + 1, uS_n^{(N)})] = kdS_n^{(N)} - Sup[f(dS_n^{(N)}), w(n + 1, dS_n^{(N)})]$$

soit :

$$k = \frac{Sup[f(uS_n^{(N)}), w(n + 1, uS_n^{(N)})] - Sup[f(dS_n^{(N)}), w(n + 1, dS_n^{(N)})]}{(u - d)S_n^{(N)}}$$

Rappelons qu'en AOA, un portefeuille sans risque doit rapporter la taux d'intérêt sans risque, ce qui donne :

$$\left[kS_n^{(N)} - w(n, S_n^{(N)}) \right] e^{rh} = kuS_n^{(N)} - Sup[f(uS_n^{(N)}), w(n + 1, uS_n^{(N)})].$$

En utilisant les deux égalités précédentes et le fait que $p = \frac{1+R-d}{u-d}$, on obtient :

$$w(n, S_n^{(N)}) = \frac{pSup[f(uS_n^{(N)}), w(n + 1, uS_n^{(N)})] + (1 - p)Sup[f(dS_n^{(N)}), w(n + 1, dS_n^{(N)})]}{1 + R}$$

On observe que le numérateur correspond à l'écriture d'une espérance sous probabilité risque-neutre :

$$\begin{aligned} w(n, S_n^{(N)}) &= \frac{\mathbb{E}_{\mathbb{Q}} \left\{ \text{Sup} \left[f(S_{n+1}^{(N)}), w(n+1, S_{n+1}^{(N)}) \right] \right\}}{1+R} \\ &= \mathbb{E}_{\mathbb{Q}} \left\{ \text{Sup} \left[\frac{1}{1+R} f(S_{n+1}^{(N)}), \frac{1}{1+R} w(n+1, S_{n+1}^{(N)}) \right] \right\} \end{aligned}$$

Considérons donc deux cas :

– Si $f(S_n^{(N)}) \geq w(n+1, S_{n+1}^{(N)})$ alors :

$$\begin{aligned} w(n, S_n^{(N)}) &= \frac{1}{1+R} \mathbb{E}_{\mathbb{Q}} \left[f(S_{n+1}^{(N)}) \right] \\ &= f(S_n^{(N)}) \end{aligned}$$

– Si $f(S_n^{(N)}) \leq w(n+1, S_{n+1}^{(N)})$ alors :

$$\begin{aligned} w(n, S_n^{(N)}) &= \frac{1}{1+R} \mathbb{E}_{\mathbb{Q}} \left[w(n+1, S_{n+1}^{(N)}) \right] \\ &= \frac{1}{1+R} \left[pw(n+1, uS_n^{(N)}) + (1-p)w(n+1, dS_n^{(N)}) \right] \end{aligned}$$

On en déduit donc que : $w(n, S_n^{(N)}) = \text{Sup} \left\{ f(S_n^{(N)}), \frac{1}{1+R} \left[pw(n+1, uS_n^{(N)}) + (1-p)w(n+1, dS_n^{(N)}) \right] \right\}$

En outre, il est clair que : $w(N, S_N^{(N)}) = f(S_N^{(N)})$

Les deux égalités précédentes permettent de conclure que $W_n = w(n, S_n^{(N)})$ où w vérifie la relation de récurrence suivante :

$$\begin{cases} w(N, x) &= f(x) \\ w(n, x) &= \text{Sup} \left\{ f(x), \frac{1}{1+R} [pw(n+1, ux) + (1-p)w(n+1, dx)] \right\} \end{cases}$$

□

2.2.3 Question c

La fonction *CalcMatPrixExercice* retourne la matrice des valeurs d'exercice possibles entre les dates 0 et N .

Le principe de l'algorithme est de stocker l'arbre du sous jacent dans la partie triangulaire supérieure d'une matrice carrée $(N+1) \times (N+1)$. On initialise avec S_0 , puis on fait évoluer le sous jacent avec les niveau up et down.

Initialisation : On définit la matrice de prix d'exercice par :

$$K := \begin{pmatrix} S_0 & 0 & \cdots & 0 \\ 0 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 \end{pmatrix}$$

Intérations :

```

Pour j=2 à (N+1)
  Pour i=1 à j
    faire  $K(i, j) = S_0 d^{i-1} u^{j-i}$ 
  FinPour
FinPour
    
```

Coût : On parcourt toute la partie supérieure de la matrice K , par conséquent on effectue $\sum_{j=2}^{N+1} j = \frac{N^2}{2}$ opérations. Le coût de l'algorithme est donc en $\mathcal{O}(N^2)$

2.2.4 Question d

La fonction *CalcMatPutAmericain* retourne la matrice des valeurs d'un Put américain entre les dates 0 et N .

Le principe de l'algorithme est de remonter l'arbre puisqu'on connaît le prix du Put américain à l'échéance (c'est la valeur intrinsèque). Pour passer de la colonne j à la colonne $j - 1$, on utilise la fonction $w(n, x)$ qui lie la valeur du Put américain entre les dates j et $j - 1$.

Initialisation : On définit la matrice triangulaire supérieure M contenant les valeurs du Put américain par :

$$M := \begin{pmatrix} 0 & \cdots & 0 & f(u^N S_0) \\ \vdots & \ddots & \vdots & \vdots \\ \vdots & \ddots & \vdots & f(u^{N-k} d^k S_0) \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & f(d^N S_0) \end{pmatrix}$$

Intérations :

```

Pour j=N à 1
  Pour i=1 à j
    on remplit la jeme colonne de la matrice de la façon suivante :
    
```

$$M := \begin{pmatrix} 0 & \cdots & 0 & w(N-1, u^j S_0) & \cdots & w(N-1, u^{N-1} S_0) & f(u^N S_0) \\ \vdots & \ddots & \vdots & \vdots & \cdots & \vdots & \vdots \\ \vdots & \ddots & \vdots & w(N-1, d^{i-1} u^{j-i} S_0) & \cdots & \vdots & \vdots \\ & & & \vdots & \cdots & \vdots & \vdots \\ & & & w(N-1, d^{i-1} S_0) & \cdots & \vdots & \vdots \\ \vdots & \ddots & \vdots & 0 & \ddots & \vdots & f(u^{N-k} d^k S_0) \\ & & & \vdots & 0 & w(N-1, d^{N-1} S_0) & \vdots \\ 0 & \cdots & 0 & 0 & 0 & 0 & f(d^N S_0) \end{pmatrix}$$

avec $w(n, x) = \text{Sup} \left\{ f(x), \frac{1}{1+R} [pw(n+1, ux) + (1-p)w(n+1, dx)] \right\}$

```

FinPour
FinPour
    
```

Coût : On parcourt toute la partie supérieure de la matrice M , par conséquent on effectue $\sum_{j=2}^{N+1} j = \frac{N^2}{2}$ opérations. Le coût de l'algorithme est donc en $\mathcal{O}(N^2)$.

2.3 Partie C

On considère maintenant une option à barrière "désactivante" : l'acheteur perd le droit d'exercice si le prix de l'actif dépasse un niveau L avant la date de maturité N .

La valeur de cette option barrière est donc trivialement :

$$V_n = \mathbb{E}_{\mathbb{Q}} \left[(1+R)^{n-N} \phi \left(S_N^{(N)} \right) \mathbf{1}_{\nu^n > N} \mid \mathcal{F}_n \right]$$

avec $\nu^n = \inf \{k \geq n : X_k > L\}$

Deux cas doivent donc être envisagés :

- Soit la barrière est atteinte avant la maturité (i.e. : $\exists n < N$ tel que : $S_n^{(N)} > L$), dans ce cas, l'option ne vaut rien (cas 1)
- Soit la barrière n'est pas atteinte ($\forall n < N$ $S_n^{(N)} < L$), dans ce cas l'option prend la valeur d'une option européenne. On se ramène alors au cas étudié en 2.1 (cas 2)

Cette rapide étude de cas permet d'écrire que $V_n = v \left(n, S_n^{(N)} \right)$ où la fonction v est définie par la relation de récurrence :

$$\begin{cases} v(N, x) &= \phi(x), \quad \forall x \leq L \\ v(n, x) &= \frac{1}{1+R} [pv(n+1, xu) + (1-p)v(n+1, xd)], \quad \forall x \leq L, \quad 0 \leq n < N \\ v(n, x) &= 0, \quad \forall x > L, \quad 0 \leq n \leq N \end{cases}$$

Les deux premières égalités correspondent au cas 2. La démonstration est analogue à 2.1.3. La dernière égalité correspond au cas 1.

De la même façon qu'en 2.2, on écrit une fonction *CalcMatUpOutPut* qui retourne à la date 0 la valeur d'une option barrière de seuil L et de payoff $\phi(x)$.

On applique le même principe que pour la valorisation du Put américain : on remonte l'arbre puisqu'on connaît le prix d'une option à barrière à l'échéance. Pour passer de la colonne j à la colonne $j-1$, on utilise la fonction $v(n, x)$ qui lie la valeur de l'option à barrière entre les dates j et $j-1$.

En parcourant toute la partie supérieure de la matrice triangulaire, on effectue $\sum_{j=2}^{N+1} j = \frac{N^2}{2}$ opérations, ce qui donne à nouveau un coût algorithmique en $\mathcal{O}(N^2)$.

ANNEXE : CODES

```

// *****
// **   Sujet : Simulations                               ***
// **   Auteurs : Bourges Mathieu                       ***
// **               Dutang Christophe                   ***
// **               Sibilleau Michael                   ***
// **   Date : mars 2007                               ***
// *****

// *****
//                               FONCTIONS
// *****

//-----
//                               Exercice 1
//-----

// entree : S0 valeur en 0 du processus d'ohrstein uhlenbeck ; n taille d'
//          echantillon
// T maturite ; alpha force de retour ; b moyenne ; sigma volatilité
// sortie : un vecteur de taille n de loi ST
function resultat = SimulSousJacent (S0,T,alpha,b,n,sigma)
    moy = S0*exp(-alpha*T)+b*(1-exp(-alpha*T));
    va = sigma^2*((1-exp(-2*alpha*T))/(2*alpha));
    resultat = grand(1,n,'nor',moy,sqrt(va)); //simule le processus d'ohrstein
    uhlenbeck de bonne esperance et bonne variance
endfunction

// entree : n taille de l'echantillon
// sortie : affiche l'histogramme de l'echantillon obtenu par SimulSousJacent
function TraceHistogram (n,bol)
    if (bol==%t) then
        x = SimulSousJacent(0,1,1,2,n,0.5);
        k = round(1+log2(n)); // regle de Sturges pour le nombre de classes
        xbas();
        histplot(k,x);
    end
endfunction

// entree : S0 valeur en 0 du processus d'ohrstein uhlenbeck ; m nb max de
//          simulation
// h pas dans le vecteur des nombres de simulation
// T maturite ; alpha force de retour ; b moyenne ; sigma volatilité
// sortie : affiche la valeur moyenne du put obtenu par Monte Carlo et l'
//          intervalle de confiance
// a 95%
function PutMonteCarlo (m,h,S0,alpha,T,b,r,K,sigma,bol)
    if (bol==%t) then
        // Monte Carlo
        abscisse = [10:h:m];
        ech = SimulSousJacent(S0,T,alpha,b,m,sigma);
        // pour chaque taille d'echantillon, on calcule le prix par la methode de
        // Monte Carlo
        for i = 1:length(abscisse)
            ech_i = ech(1:abscisse(i));
            moyenne = mean(exp(-r*T)*max(K-ech_i,0));
            ecart_type = st_deviation(exp(-r*T)*max(K-ech_i,0));
            down = moyenne - (1.96*ecart_type)/(sqrt(abscisse(i)));
            up = moyenne + (1.96*ecart_type)/(sqrt(abscisse(i)));
            IC(:,i) = [up ; moyenne ; down];
        end

        // valeur theorique
        moy = K - S0*exp(-alpha*T)-b*(1-exp(-alpha*T));
        var = sigma^2*((1-exp(-2*alpha*T))/(2*alpha));
        ech_th = (sqrt(var)/sqrt(2*%pi))*exp(-moy^2/(2*var))+moy*(1-cdfnor("PQ",-
            moy/(sqrt(var)),0,1));
        // affichage
        xbas();
        plot2d(abscisse,ech_th*ones(1,length(abscisse)),style=[4]);
    end
endfunction

```

```

        plot2d(abscisse,IC(1,:),style=[1]);
        plot2d(abscisse,IC(2,:),style=[2]);
        plot2d(abscisse,IC(3,:),style=[1]);

        disp("Valeur du Put par Monte Carlo");
        disp(moyenne);
        disp("Valeur du Put par Black & Scholes");
        disp(ech_th);
    end
endfunction

// entree : S0 valeur en 0 du processus d'ohrstein uhlenbeck ; n taille d'
// echantillon
// T maturite ; alpha force de retour ; b moyenne ; sigma volatilité ; t vecteur
// des temps
// sortie : une matrice de taille n x length(t) ( les dates sont les colonnes ;
// les differents proc sont les lignes)
// contenant les differents processus simules d'ohrstein ulhenbeck aux
// differentes dates sans approximation
function resultat = SimulSousJacentTrajExact (n,t,S0,alpha,b,sigma)
    res = SimulSousJacent(S0,t(1),alpha,b,n,sigma)';
    //simulation du processus entre chaque periode de temps
    for( i=2:(length(t)))
        moyenne = exp(alpha*(t(i-1)-t(i)))*res(:,i-1)+b*(1-exp(alpha*(t(i-1)-t(i)
            )));
        va = sigma^2/(2*alpha)*(1-exp(2*alpha*(t(i-1)-t(i))));
        temp=grand(n,1,'nor',0,1)*sqrt(va)+moyenne;
        res = [res temp];
    end
    resultat = res;
endfunction

// entree : S0 valeur en 0 du processus d'ohrstein uhlenbeck ; n taille d'
// echantillon
// T maturite ; alpha force de retour ; b moyenne ; sigma volatilité ; t vecteur
// des temps
// sortie : une matrice de taille n x length(t) ( les dates sont les colonnes ;
// les differents proc sont les lignes)
// contenant les differents processus simules d'ohrstein ulhenbeck aux
// differentes dates par la methode d'Euler
function resultat = SimulSousJacentTrajEuler (n,t,S0,alpha,b,sigma)
    res = SimulSousJacent(S0,t(1),alpha,b,n,sigma)';
    // approximation d'euler entre chaque periode de temps
    for i=2:(length(t))
        temp2 = grand(n,1,'nor',0,1)*sigma*sqrt(t(i)-t(i-1));
        res(:,i) = res(:,i-1)+alpha*(t(i)-t(i-1))*(b-res(:,i-1))+temp2;
    end;
    resultat = res;
endfunction

// entree : numero de la methode 0 pour exacte 1 pour Euler
// sortie : affiche la trajectoire simulee par les fonctions precedentes
function TraceTrajectoire(t,S0,alpha,b,sigma,bol,methode)
    if (bol==%t)
        if (methode==0) then
            y = SimulSousJacentTrajExact (1,t,S0,alpha,b,sigma);
        else
            y = SimulSousJacentTrajEuler (1,t,S0,alpha,b,sigma);
        end
        xbas();
        plot2d(t,y,style=[1]);
    end
endfunction

// entree : S0 valeur en 0 du processus d'ohrstein uhlenbeck ; m nb max de
// simulation
// h pas dans le vecteur des nombres de simulation
// T maturite ; alpha force de retour ; b moyenne ; sigma volatilité ; t vecteur
// des temps
// sortie : affiche la fonction de repartition empirique et l'histogramme pour

```

```

// les differents processus simules d'ohrstein ulhenbeck aux differentes dates
// par la methode d'Euler
function TraceFdrHisto (n,t,S0,alpha,b,sigma,bol)
    if (bol==%t) then
        T=t(length(t));
        x = SimulSousJacentTrajEuler (n,t,S0,alpha,b,sigma);
        x = gsort(x(:,size(x,2)), 'r', 'i');
        k = round(1+log2(n));
        moyenne=S0*exp(-alpha*T)+b*(1-exp(-alpha*T));
        st =sqrt(sigma^2*((1-exp(-2*alpha*T))/(2*alpha)));
        y=(1/n):(1/n):(1-1/n);
        z=cdfnorf("X",moyenne*ones(1,n-1),st*ones(1,n-1),y,1-y);
        //affichage
        xbasf();
        subplot(1,2,1);
        plot2d2(x(1:n-1),y,style=2);
        plot2d2(z,y,style=3);
        subplot(1,2,2);
        histplot(k,x);
    end
endfunction

// entree : S0 valeur en 0 du processus d'ohrstein uhlenbeck ; n taille d'
// echantillon
// T maturite ; alpha force de retour ; b moyenne ; sigma volatilité ; t vecteur
// des temps
// sortie : affiche la valeur moyenne du put obtenu et l'intervalle de confiance
// a 95% pour les processus simules d'ohrstein ulhenbeck aux differentes dates
// par la methode d'Euler
function PutEuler (m,h,t,S0,alpha,b,r,K,sigma,bol)
    if (bol==%t) then
        // on suppose que le vecteur t commence par le temps courant et qu'il
        // finit par T
        T=t(length(t));
        abscisse = [10:h:m];
        simulations=SimulSousJacentTrajEuler(m,t,S0,alpha,b,sigma);
        Simul_ST=simulations(:,length(t));
        // pour chaque taille d'echantillon, on calcule le prix par la methode d'
        // Euler
        for i=1:length(abscisse)
            Puts=exp(-r*T)*max(K-Simul_ST(1:abscisse(i)),0);
            moyenne = mean(Puts);
            ecart_type = st_deviation(Puts);
            down = moyenne - (1.96*ecart_type)/(sqrt(abscisse(i)));
            up = moyenne + (1.96*ecart_type)/(sqrt(abscisse(i)));
            IC(:,i) = [up ; moyenne ; down];
        end

        // valeur theorique
        moy = K - S0*exp(-alpha*t(length(t)))-b*(1-exp(-alpha*t(length(t))));
        var = sigma^2*((1-exp(-2*alpha*t(length(t))))/(2*alpha));
        ech_th = (sqrt(var)/sqrt(2*pi))*exp(-moy^2/(2*var))+moy*(1-cdfnorf("PQ",-
            moy/(sqrt(var)),0,1));
        xbasf();
        plot2d(abscisse,ech_th*ones(1,length(abscisse)),style=4);
        plot2d(abscisse,IC(1,:),style=1);
        plot2d(abscisse,IC(2,:),style=2);
        plot2d(abscisse,IC(3,:),style=1);

        disp("Valeur du Put par Euler");
        disp(moyenne);
        disp("Valeur du Put par Black & Scholes");
        disp(ech_th);
    end
endfunction

//-----
// Exercice 2

```

```

//-----
// entree : N nombre de sous periode ; sigma volatilité ; T maturité ;
// r taux sans risque ; S0 point initial ; K prix d'exercice ;
// u niveau up ; d niveau down ; p probabilité du niveau up
// R force d'actualisation ; h pas
// sortie : renvoie le prix du put calculé par la méthode binomiale
function resultat=PutCombinatoire(N,sigma,T,r,S0,K,u,d,p,R,h,bol)
resultat=0;
if (bol==%t) then
    somme=0;
    //pour chaque sous periode calcul du prix du put
    for i=0:N
        comb=TabFact(N+1)/(TabFact(i+1)*TabFact(N-i+1));
        VT=exp(-r*T)*max(0,K-(S0*u^i*d^(N-i)));
        somme=somme+comb*(p^i)*((1-p)^(N-i))*VT;
    end;
    resultat=somme;
end;
endfunction

// entree : N taille de l'arbre ; sigma volatilité ; T maturité ;
// r taux sans risque ; S0 point initial ; K prix d'exercice ;
// u niveau up ; d niveau down ; p probabilité du niveau up
// R force d'actualisation ; h pas
// sortie : renvoie le prix du put calculé par la méthode des arbres
function resultat=PutArbre(N,sigma,T,r,S0,K,u,d,p,R,h,bol)
resultat=0;
if (bol==%t) then
    for i=0:N
        ST=S0*(u^(N-i))*(d^i);
        V(i+1)=max(0,K-ST);
    end;
    //remontée de l'arbre
    for i=0:(N-1)
        aux=zeros(length(V)-1,1);
        for j=1:(length(aux))
            aux(j)=(p*V(j)+(1-p)*V(j+1))/(1+R);
        end;
        V=aux;
    end;
    resultat=V;
end;
endfunction

// entree : sigma volatilité ; T maturité ;
// r taux sans risque ; S0 point initial ; K prix d'exercice ;
// sortie : renvoie le prix du put calculé par Black & Scholes
function resultat=PutTheorique(sigma,T,r,S0,K,bol)
resultat=0;
if (bol==%t) then
    d0=log(K/(S0*exp(r*T)))/(sigma*sqrt(T))-sigma*sqrt(T)/2;
    d1=d0+sigma*sqrt(T);
    resultat=K*exp(-r*T)*cdfnor("PQ",d1,0,1)-S0*cdfnor("PQ",d0,0,1);
end;
endfunction

// entree : N nombre de sous periodes ; sigma volatilité ; T maturité ;
// r taux sans risque ; S0 point initial ; K prix d'exercice ;
// u niveau up ; d niveau down ; p probabilité du niveau up
// R force d'actualisation ; h pas
// sortie : affiche la valeur du put théorique et approximée en fonction du point
// initial
function TraceValPut(N,sigma,T,r,S0,K,u,d,p,R,h,bol,methode)
if (bol==%t) then
    x=[0.01:0.1:2];
    y1=zeros(1,length(x));
    y2=zeros(1,length(x));
    BS=zeros(1,length(x));

```

```

//calcul pour les differentes methodes
for i=1:length(x)
    y1(i)=PutCombinatoire(N,sigma,T,r,x(i),K,u,d,p,R,h,%t); // question b
    y2(i)=PutArbre(N,sigma,T,r,x(i),K,u,d,p,R,h,%t); // question c
    BS(i)=PutTheorique(sigma,T,r,x(i),K,bol);
end;

xbasc();
if(methode == 1) then
    plot2d(x,y1,-1);
    plot2d(x,BS,2);
else
    plot2d(x,y2,-2);
    plot2d(x,BS,2);
end
end;
endfunction

// entree : N nombre de sous periode ; sigma volatilité ; T maturité ;
// r taux sans risque ; S0 point initial ; K prix d'exercice ;
// u niveau up ; d niveau down ; p probabilité du niveau up
// R force d'actualisation ; h pas
// sortie : renvoie le prix du call calcule par la methode binomiale
function resultat=CallCombinatoire(N,sigma,T,r,S0,K,u,d,p,R,h,bol)
resultat=0;
if (bol==%t) then
    somme=0;
    for i=0:N
        comb=TabFact(N+1)/(TabFact(i+1)*TabFact(N-i+1));
        ST=S0*u^i*d^(N-i);
        somme=somme+comb*p^i*(1-p)^(N-i)*(exp(-r*T)*max(0,ST-K));
    end;
    resultat=somme;
end;
endfunction

// entree : N nombre de sous periode ; sigma volatilité ; T maturité ;
// r taux sans risque ; S0 point initial ; K prix d'exercice ;
// u niveau up ; d niveau down ; p probabilité du niveau up
// R force d'actualisation ; h pas
// sortie : trace l'erreur en valeur absolue sur la relation de parité call/put
function VerifParite(N,sigma,T,r,K,u,d,p,R,h,bol)
if (bol==%t) then
    x=0.01:0.1:10
    for i=1:length(x)
        y(i)=CallCombinatoire(N,sigma,T,r,x(i),K,u,d,p,R,h,%t)-PutCombinatoire(N,
            sigma,T,r,x(i),K,u,d,p,R,h,%t);
        z(i)=x(i)-K*exp(-r*T);
    end;
    xbasc();
    plot2d(x,abs(y-z)); // affichage de l'erreur
end;
endfunction

// entree : S0 point initial ; N taille d'arbre
// u niveau up ; d niveau down ; p probabilité du niveau up
// sortie : renvoie la matrice des prix d'exercice pour le put americain
function Matrice_S=CalcMatPrixExercice(S0,u,d,N,bol)
Matrice_S=zeros(N+1,N+1);
if (bol==%t) then
    Matrice_S(1,1)=S0;
    for i=2:(N+1) // i colonne
        for j=1:i // j ligne
            Matrice_S(j,i)=S0*d^(j-1)*u^(i-j);
        end;
    end;
end;
endfunction

// entree : S0 point initial ; N taille d'arbre

```

```

// u niveau up ; d niveau down ; p probabilité du niveau up
// R force d'actualisation; K prix d'exercice
// sortie : renvoie la matrice des valeurs du put américain (par la méthode des
// arbres)
function Matrice_V=CalcMatPutAmericain(S0,u,d,N,R,K,bol)
Matrice_V=zeros(N+1,N+1);
if (bol==%t) then
    Matrice_S=CalcMatPrixExercice(S0,u,d,N,%t);
    Matrice_V(:,N+1) = max(K-Matrice_S(:,N+1),0);
    p = (1+R-d)/(u-d);
    for i=N:-1:1
        for j = 1:i
            Matrice_V(j,i) = max(max(K-Matrice_S(j,i),0), (p*Matrice_V(j,i+1)+(1-p)*
                Matrice_V(j+1,i+1))/(1+R)) ;
        end
    end
end;
endfunction

// entree : S0 point initial ; N taille d'arbre
// u niveau up ; d niveau down ; p probabilité du niveau up
// R force d'actualisation; K prix d'exercice
// sortie : renvoie la matrice des valeurs du Up Out Put (par la méthode des
// arbres)
function Valeur=CalcMatUpOutPut(S0,u,d,N,R,K,L,bol)
Valeur=0;
if (bol==%t) then
    Matrice_V=zeros(N+1,N+1);
    Matrice_S=CalcMatPrixExercice(S0,u,d,N,%t);
    Matrice_V(:,N+1) = (Matrice_S(:,N+1)<=L).*max(K-Matrice_S(:,N+1),0);
    p = (1+R-d)/(u-d);
    //remontée de l'arbre
    for i=N:-1:1
        for j = 1:i
            Matrice_V(j,i) = (Matrice_S(j,i)<=L)*(p*Matrice_V(j,i+1)+(1-p)*Matrice_V(
                j+1,i+1))/(1+R);
        end
    end
    Valeur=Matrice_V(1,1);
end;
endfunction

// entree : N entier maximal dont on veut calculer la factorielle
// sortie : stocke les différentes factorielles de 0 à N+1 dans TabFact
function InitTabFact(N)
    global TabFact
    TabFact = ones(1:(N+1))
    for(i = 2:(N+1))
        TabFact(i) = TabFact(i-1)*(i-1);
    end
endfunction

```

```

// *****
// **   Sujet : Simulations           ***
// **   Auteurs : Bourges Mathieu    ***
// **           Dutang Christophe     ***
// **           Sibilleau Michael     ***
// **                                     ***
// **   Date : mars 2007              ***
// *****

// *****
//                                     MAIN
// *****

// definir chemin comme repertoire de travail
chemin = 'C:\Documents and Settings\voreppe\Bureau';
chdir(chemin);
getf("fonctionTD2.sci"); // recup des fonctions

// entree : "code" de la question
// 1 => question 1B
// 2 => question 1D
// 3 => question 1E
// 4 => question 1F
// 5 => question 1f
// 6 => question 2Ab
// 7 => question 2Ac
// 8 => question 2Ad
// 9 => question 2Ae
// 10 => question 2Af
// 11 => question 2B
// 12 => question 2C
// sortie : la quesiton demandee

function main(question)
    global TabFact
    TabFact=zeros(1);

    select question
// -----
    case 1 then // Question 1B
        TraceHistogram(100000,%t);
// -----
    case 2 then // Question 1D
        S0=0;
        alpha=1;
        T=1;
        b=2;
        sigma=0.5;
        r=0.025;
        t=0:0.005:1;
        K=1;
        m = 10000;
        h = 100;
        PutMonteCarlo(m,h,S0,alpha,T,b,r,K,sigma,%t);
// -----
    case 3 then // Question 1E
        S0=0;
        alpha=1;
        T=1;
        b=2;
        sigma=0.5;
        r=0.025;
        t=0.1:0.1:100;
        K=1;
        TraceTrajectoire(t,S0,alpha,b,sigma,%t,0);
// -----
    case 4 then // Question 1F
        S0=0;
        alpha=1;

```

```

T=1;
b=2;
sigma=0.5;
r=0.025;
t=0:.5:200;
K=1;
n = 1000;
TraceFdrHisto (n,t,S0,alpha,b,sigma,%t);
// -----
case 5 then          // Question 1F
S0=0;
alpha=1;
T=1;
b=2;
sigma=0.5;
r=0.025;
t=0:0.005:1;
K=1;
m = 5000;
h = 500;
PutEuler (m,h,t,S0,alpha,b,r,K,sigma,%t);
// -----
case 6 then          // Question 2Ab
N=100;
if(length(TabFact) <> N+1)
    InitTabFact(N);
end
sigma=0.3;
T=1;
r=0.05;
S0=1;
K=1;
h=T/N;
R=exp(r*h)-1;
u=exp(sigma*sqrt(h));
d=exp(-sigma*sqrt(h));
p=(exp(r*h)-d)/(u-d);
disp("Valeur du Put par la methode binomiale");
disp(PutCombinatoire(N,sigma,T,r,S0,K,u,d,p,R,h,%t));
// -----
case 7 then          // Question 2Ac
N=100;
if(length(TabFact) <> N+1)
    InitTabFact(N);
end
sigma=0.3;
T=1;
r=0.05;
S0=1;
K=1;
h=T/N;
R=exp(r*h)-1;
u=exp(sigma*sqrt(h));
d=exp(-sigma*sqrt(h));
p=(exp(r*h)-d)/(u-d);
disp("Valeur du Put par les arbres");
disp(PutArbre(N,sigma,T,r,S0,K,u,d,p,R,h,%t));
// -----
case 8 then          // Question 2Ad
N=100;
if(length(TabFact) <> N+1)
    InitTabFact(N);
end
sigma=0.3;
T=1;
r=0.05;
S0=1;
K=1;
h=T/N;
R=exp(r*h)-1;

```

```

    u=exp(sigma*sqrt(h));
    d=exp(-sigma*sqrt(h));
    p=(exp(r*h)-d)/(u-d);
    methode = 2 ; // pour choisir la methode de valorisation : 1
                  methode binomiale et 2 methode des arbres
    TraceValPut(N,sigma,T,r,S0,K,u,d,p,R,h,%t,methode);
// -----
case 9 then // Question 2Ae
    N=100;
    if(length(TabFact) <> N+1)
        InitTabFact(N);
    end
    sigma=0.3;
    T=1;
    r=0.05;
    S0=1;
    K=1;
    h=T/N;
    R=exp(r*h)-1;
    u=exp(sigma*sqrt(h));
    d=exp(-sigma*sqrt(h));
    p=(exp(r*h)-d)/(u-d);
    VerifParite(N,sigma,T,r,K,u,d,p,R,h,%t);
// -----
case 10 then // Question 2Af
    N=100;
    if(length(TabFact) <> N+1)
        InitTabFact(N);
    end
    sigma=0.3;
    T=1;
    r=0.05;
    S0=1;
    K=1;
    h=T/N;
    R=exp(r*h)-1;
    u=exp((r-(sigma^2)/2)*h+sigma*sqrt(h));
    d=exp((r-(sigma^2)/2)*h-sigma*sqrt(h));
    p=0.5;
    methode = 2 ; // pour choisir la methode de valorisation : 1
                  methode binomiale et 2 methode des arbres
    TraceValPut(N,sigma,T,r,S0,K,u,d,p,R,h,%t,methode);
    disp("pause volontaire, taper return pour continuer");
    pause
    VerifParite(N,sigma,T,r,K,u,d,p,R,h,%t);
// -----
case 11 then // Question 2B
    N=100;
    if(length(TabFact) <> N+1)
        InitTabFact(N);
    end
    sigma=0.3;
    T=1;
    r=0.05;
    S0=1;
    K=1;
    h=T/N;
    R=exp(r*h)-1;
    u=exp(sigma*sqrt(h));
    d=exp(-sigma*sqrt(h));
    p=(exp(r*h)-d)/(u-d);
    res=CalcMatPutAmericain(S0,u,d,N,R,K,%t);
    disp("valeur du Put americain : ");
    disp(res(1,1)); // prix du Put americain
    disp("valeur théorique du put européen :")
    disp(PutTheorique(sigma,T,r,S0,K,%t));
// -----
case 12 then // Question 2C
    N=100;
    if(length(TabFact) <> N+1)

```

```
        InitTabFact(N);
    end
    sigma=0.3;
    T=1;
    r=0.05;
    S0=1;
    K=1;
    h=T/N;
    R=exp(r*h)-1;
    u=exp(sigma*sqrt(h));
    d=exp(-sigma*sqrt(h));
    p=(exp(r*h)-d)/(u-d);
    L = 1.5;
    res=CalcMatUpOutPut(S0,u,d,N,R,K,L,%t);
    disp("valeur du Up Out Put : ");
    disp(L,"barriere de ")
    disp(res(1,1)); // prix du Up Out Put
    disp("valeur théorique du put européen :")
    disp(PutTheorique(sigma,T,r,S0,K,%t));
end
endfunction

main(4)
```